πBox: A Platform for Privacy-Preserving Apps

Sangmin Lee, Edmund L. Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov

The University of Texas at Austin

Abstract

We present πBox, a new application platform that prevents apps from misusing information about their users. To strike a useful balance between users' privacy and apps' functional needs, πBox shifts much of the responsibility for protecting privacy from the app and its users to the platform itself. To achieve this, πBox deploys (1) a sandbox that spans the user's device and the cloud, (2) specialized storage and communication channels that enable common app functionalities, and (3) an adaptation of recent theoretical algorithms for differential privacy under continual observation. We describe a prototype implementation of πBox and show how it enables a wide range of useful apps with minimal performance overhead and without sacrificing user privacy.

1 Introduction

On mobile platforms such as iOS and Android, Web browsers such as Google Chrome, and even smart televisions such as Google TV or Roku, hundreds of thousands of software apps provide services to users. Their functionality often requires access to potentially sensitive user data (e.g., contact lists, passwords, photos), sensor inputs (e.g., camera, microphone, GPS), and/or information about user behavior.

Most apps use this data responsibly, but there has also been evidence of privacy violations [2, 36, 43, 54, 56]. Corporations often restrict what apps employees can install on their phones to prevent an untrusted app—or a cloud provider that an app communicates with—from leaking proprietary information [11, 28].

There is an inherent trade-off between users' privacy and apps' functionality. An app with no access to user data (e.g., one running in Native Client [39]) cannot leak anything sensitive, but many apps cannot function without such data. For example, a password management app needs access to passwords, an audio transcription app needs access to the recordings of user's speech, etc.

Existing confinement mechanisms deployed on platforms such as iOS and Android rely on users to explicitly grant permissions to apps. In theory, users can decide how much privacy to sacrifice for functionality. In practice, permissions are very coarse-grained (e.g., an app that has permission to access the network can send out whatever it wishes to whomever it wishes), and apps often request more permissions than they need [19, 25] and use granted permissions in unexpected ways (e.g., an app with permission to show the user's location on a map may transmit this location to other parties). Users—who are inundated with permission requests and may not fully understand the implications—often blindly grant all requests [20] or even disable notifications [37], implicitly entrusting all apps with their private data.

Our contributions. This paper describes πBox, a new platform for confining untrusted apps that balances apps' functional needs against their users' privacy, largely preserving both. To achieve this balance, πBox isolates each user's instance of an app from the other instances and users, and only allows communication through a few well-defined channels whose functionality meets the needs of many apps. Because these channels are controlled by πBox, πBox can give rigorous privacy guarantees about the information that flows through them.

The key idea behind πBox is to shift much of the responsibility for protecting user privacy from the apps to the platform. We use three novel technical mechanisms:

1. A sandbox that spans a user's device and a cloud back-end. The latter may be supplied by the device's platform provider (e.g., Apple or Google) or another entity (e.g., the user's employer).
2. Five specialized storage and communications systems that enable a variety of apps to do useful work within πBox while preserving user privacy.
3. An adaptation and implementation of *differential privacy under continual observation* that improves the trade-off between accuracy and privacy of released statistics (e.g., ad impression counts).

Because πBox's sandbox spans the device and the cloud, πBox can help enterprises deploy **bring-your-own-app** (BYOA) policies that allow users to execute apps from untrusted publishers on a trusted platform. This platform may run on the premises under the enterprise's direct control or be part of an external "app store" or hosting infrastructure. Similar to bring-your-own-device (BYOD) policies, where companies install profiles and security software on employee-owned devices used for work, a company might restrict apps to run only within πBox, thus ensuring that these apps—and any information they access—are securely confined.

This paper addresses three research questions raised by this architecture. Can we construct useful apps under these constraints? Can we adapt differentially private aggregation to an environment where app providers need to query periodically updated statistics of user activities? Are the overheads of πBox acceptable?

To answer these questions, we constructed (1) a prototype of πBox and (2) a set of sample apps that represent common app types and demonstrate the utility of our platform: a cloud-backed password vault, an ad-supported news reader, and a transcription service. We also ported two open-source Android apps: the OsmAnd navigation app [41] and ServeStream, an HTTP-streaming media player and media server browser [51]. In Section 2.5, we explain in more detail the classes of apps and app features supported by πBox.

πBox uses differential privacy to prevent aggregate statistics from leaking too much information about users to app publishers. Conventional differentially private queries on static datasets can be very inaccurate when the input data is changing due to user behavior. Instead, we apply algorithms for differential privacy under continual observation [16]—in particular, delayed-output counters. We also list the parameters that enable an app publisher to tune the amount, frequency, and/or accuracy of the reported statistics subject to the platform's bound on the rate of information leakage. The resulting relative error rates on real-world traces are five times lower than with conventional differentially private counters.

The paper proceeds as follows. Section 2 presents an overview of πBox's design. Section 3 shows how πBox deploys differential privacy under continual observation and privacy-preserving top-K lists to implement aggregate channels. Section 4 describes our prototype implementation. Section 5 evaluates it and describes the apps we developed or ported to πBox. Section 6 discusses related work. Section 7 concludes.

2 Design

πBox is a platform for executing apps and associated remote services. There are three types of principals involved in πBox: (1) the platform *provider* who supplies the client (either software, e.g., Google Chrome, or both hardware and software, e.g., Apple iPhone, Google Nexus 7, or Kindle Fire), as well as the cloud resources on which app instances execute, and deploys πBox on both the client and the cloud; (2) *users* who invoke and use untrusted apps on their local devices and their slice of the cloud; and (3) *publishers* who provide apps, content for apps, and/or advertisements.

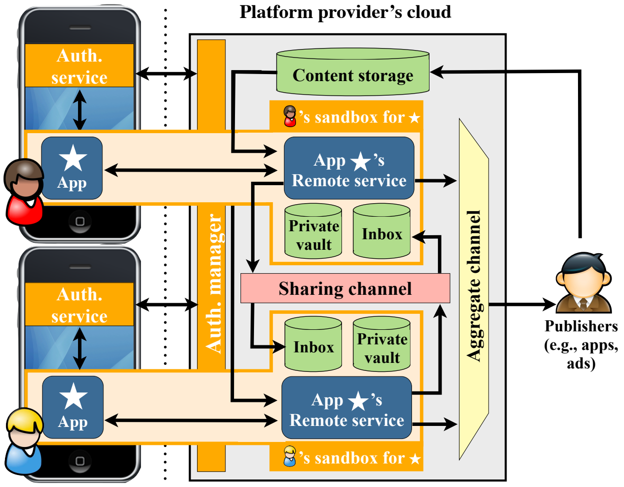


FIGURE 1—Architecture of πBox.

2.1 Threat model

πBox is based on the following design philosophy: *do not trust the apps nor rely on the users to make fine-grained privacy decisions; instead, trust the platform to enforce privacy*. We argue that trusting the platform provider is far more reasonable than expecting users to judge the trustworthiness of many different, often obscure app publishers. After all, users must already trust the platform provider to not leak their private data. Furthermore, third-party platform providers are often trusted brands such as Google, Apple, and Amazon that have strong incentives to take care of their customers' data. Therefore, we assume that both users and app publishers trust the platform, but users do not trust the publishers. Furthermore, we neither assume that the provider trusts the publishers, nor rely on auditing by the provider to eliminate misbehaving apps.[1]

πBox is thus designed for the scenario where *an untrusted app runs in a trusted sandbox*. In this model, the app's publisher may be malicious, the code of the app may attempt to leak users' private data or reveal information about its users to the publisher, some of the app's users may be colluding with the app in an attempt to learn other users' data, etc. That said, the attacker is subject to

[1] Platforms that do audit apps such as Google Play provide additional assurance that is complementary to what πBox provides.

standard computational feasibility constraints (e.g., the attacker cannot subvert cryptographic primitives).

The sandbox provided by πBox is assumed to be trusted. This includes both the components running on the client device and those running in the cloud. Like any software, if πBox is implemented incorrectly, it may be subject to code injection and other attacks that compromise the "ideal sandbox" abstraction. These attacks are outside the scope of this paper, which focuses primarily on the design of the sandbox. Another way in which the "ideal sandbox" abstraction may be violated is via covert (e.g., timing) channels between processes running in the sandbox and those outside the sandbox [33, 47]. If an implementation of πBox is vulnerable to such channels, apps may be able to exfiltrate private data.

There has been much research on sandboxing mechanisms (e.g., [27, 31, 60], among others). This work is orthogonal and complementary to the design of πBox and can be applied to any implementation thereof.

2.2 Extended sandbox

Apps in πBox have two halves: one runs locally on the user's device, the other (optional) runs remotely in the cloud. πBox, executing as the platform both on the device and in the cloud,[2] supplies a per-user, per-app sandbox that spans the device and the cloud. In effect, πBox provides the abstraction that a slice of the cloud is part of the user's device: all of the app's computations and storage are done within this "distributed" device, which is otherwise isolated to protect the user's privacy.

The local half of an app running on the user's device can only connect to the remote half associated with the same app and user. The local half does so by making a request to the *authentication service* running as part of the platform on the device. This service sends the user's credentials and the app's ID to the *authentication manager* running as a part of the platform in the cloud (see Figure 1). Upon successful authentication, the authentication manager starts up the requesting app's remote half for that specific user and opens a secure channel between the local and remote halves.

2.3 Storage and communication

An app running within πBox cannot write data or establish network connections outside of the sandbox. To support app functionality, πBox provides **five restricted storage and communication channels** (see Table 1).

The *private vault* provides per-sandbox (i.e., per-user, per-app) storage that lets an app instance store data specific to a particular user (e.g., user profile, location, query history, etc.) in order to provide personalized services. For example, a password app may use the vault to store the user's passwords, while a news reader app may store keywords of the articles the user has read. Each sandboxed app instance has read/write access to its own private vault; no one else has any access rights.

	Written by	Read by	Purpose
Shared channels for all users of an app			
Content storage	Publisher	App	Store app data and content
Aggregate channel	App	Publisher	Collect usage statistics
Individual channels for each app instance			
Private vault	App	App	App-specific
Inbox	App (via sharing channel), Publisher	App	Receive shared content, notifications from publisher
Sharing channel	App	App (via inbox)	Share content

TABLE 1—Channels in πBox.

The *content storage* provides per-publisher storage for the content that app instances need to function, e.g., maps for a navigation app. Each publisher has read/write access to its own content storage so that the publisher can (1) update the content and (2) grant read-only access to apps that need this content. Apps may draw content from multiple publishers' content storage. For example, an ad-supported news reader may load news articles from a news publisher's storage and ads from an ad broker's storage. Although content storage is shared across all sandboxes that have access to it, read-only access prevents communication between app instances.

The *aggregate channel* provides a per-app channel (shared among all instances of an app) for publishers to collect statistics on users' collective behavior while protecting privacy of individual users. For example, publishers of advertising-supported apps may collect the total number of ad impressions, but not which user viewed which ad. Similarly, publishers of news or video streaming apps may learn which articles or videos are popular, but not who viewed what content. Publishers have read access to their respective aggregate channels, and each app has write access to its channel. In Section 3, we describe how πBox employs differential privacy to protect data released via this channel.

The *inbox* provides per-sandbox storage for the user of a particular app instance to receive information from the app's publisher as well as the content, if any, shared by other users of the same app. Each sandbox has read/write access to its inbox. All writes from the publisher or other users must go through πBox; when publishers want to

[2] Apple (iOS/iCloud) and Google (Android/Cloud Services) already provide app platforms that extend from users' devices to the cloud.

communicate with their apps' users, they submit messages with the user as the recipient, and πBox delivers the message to the appropriate inbox.

Finally, the *sharing channel* provides a per-sandbox method for sharing content with other users of the same app. To ensure that all recipients of the shared content are explicitly approved by the user, we rely on a trusted, platform-controlled dialog box (similar to a "powerbox," which is traditionally used to restrict the paths an app can access [34, 50]). When a user wants to share content from an app, the app writes the data to be shared into its own sharing channel (to which no other sandbox has access) and notifies the platform. πBox controls the rest of the sharing process: it (1) reads in the data, (2) presents the data to the user in a dialog box that explicitly notifies the user about the imminent sharing of the presented data, (3) prompts the user to confirm the recipients, and, upon confirmation, (4) writes the shared content to the inboxes of the designated recipients' sandboxes. This design ensures that users are aware when and with whom sharing occurs, but it cannot prevent the app from surreptitiously leaking private information in the shared data (e.g., through steganography).

2.4 Advertising and third-party services

Advertising. To broadly support free apps, many of which are financed by ads, πBox must support in-app advertising. Traditionally, advertisers tell ad networks which ads to display, how much they are willing to pay per impression, and the interests they are targeting. Ad networks organize ads into lists ranked by factors such as the bid, number of impressions already made, etc. When an app wants to display an ad, the ad network provides an ad based on the user's perceived interests.

To prevent apps from leaking users' private data to advertisers, πBox changes this process: (1) the ad network must store its ads in content storage on the πBox cloud platform, (2) the number of impressions must be released via the aggregate channel (see Section 3.1), and (3) the logic for selecting and fetching an ad from content storage (based on the user's profile, activities, etc.) and the logic for outputting to the aggregate channel must be implemented inside the app (e.g., as part of a SDK or library) and executed inside the sandbox. For efficiency, πBox allows publishers to share content storage across multiple apps. Since apps have read-only access, this does not affect privacy guarantees.

πBox protects users' identities and thus prevents ad networks from singling out individuals who may be engaged in ad impression/click fraud. That said, other defenses [22]—per-user thresholds on the number of impressions/clicks, bait ads, and using historical statistics to detect apps that pad the number of impressions/clicks—continue to be effective even with πBox.

Ads that click-through to external sites can leak a user's identity (or at least the IP address) and other private information.[3] In πBox, arbitrary network traffic out of the sandbox is not allowed, and click-through ads must redirect the user to trusted platform resources, e.g., an ad page in the ad network's content storage.

Although not yet implemented, conventional click-through ads can be supported in πBox with some modifications. First, all click-through URLs must be prespecified and static for all app instances (they cannot be dynamically generated or otherwise based on the information observed by a given instance). This still allows a potential leak because the app's choice of predefined ads to show to the user may depend on the user's private information, but requiring static URLs limits the rate of leakage. Second, the platform must verify that the click indeed originated from the user. To support this, πBox can use a trusted powerbox dialog to prompt the user for explicit consent, similar to the sharing channel, before permitting the click to go through. We believe, however, that this point in the design space for ad support sacrifices privacy, complicates the guarantees provided by πBox, and forces users to make privacy decisions for which they may not fully understand the implications.

πBox does not currently support ad networks that choose which ads to serve via a real-time auction. Such auctions require either that users' profiles be sent to the advertisers (so they know what they are bidding on), or that all bidding logic be part of the sandbox. Alternatively, there exist proposals for privacy-preserving ad auctions [46]. Advertising based on real-time bidding accounts for less than 30% of all advertising sales [45], and the introduction of "Do Not Track" in Web browsers may adversely impact auction-based advertising [17].

Third-party services. Because πBox does not allow apps to communicate outside of the platform, apps cannot use external third-party services such as content delivery networks (CDNs). As with ads, apps running on πBox can only access content and use services that are hosted by the platform provider and published in the read-only content storage. Fortunately, many platform providers already provide services for apps, e.g., maps from Apple, Google, and Bing, or CDN services such as Amazon CloudFront and Google PageSpeed.

2.5 Apps supported by πBox

Figure 2 lists many app features and indicates whether and how πBox protects user privacy for each of them. In general, apps that do not involve sharing between

[3] For example, a set of ads may only be shown to (and thus clicked by) users matching certain criteria or even maliciously micro-targeted to specific individuals [30].

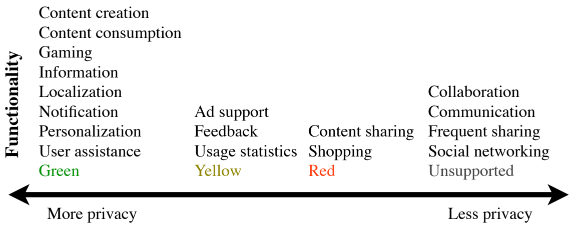


FIGURE 2—πBox support for different app features.

users are well-suited for πBox. This includes, for example, multimedia, reference, weather, and utility apps, many of which handle sensitive data (e.g., navigation, personal finance, password management, malware detection, speech recognition, etc.). πBox supports the reporting of usage statistics, user feedback, and ad impressions.

Some apps only share content occasionally, e.g., games that let users share their scores, or camera apps that let users share some of their pictures. For these apps, πBox protects user privacy with respect to the app's core functionality. Furthermore, the πBox sharing channel ensures that any content sharing is explicitly authorized by the user (malicious apps may still exfiltrate sensitive data by hiding it in shared content—see Section 2.6).

Finally, there are apps—e.g., Facebook, Twitter, or multiplayer online games—whose sole purpose is to allow users to connect, communicate, collaborate, and share content with other users. Users of such apps already expect to lose some of their privacy, and πBox can guarantee relatively little for them.

Each πBox-supported app is assigned a **privacy rating** determined by the channels it uses. Apps that only use the private vault, content storage, or inbox are **green**: they never export any data from the sandbox and cannot leak anything. Apps that use the aggregate channel are **yellow**: they may release differentially private statistics but there is a provable bound on the amount of information leaked. Finally, apps that use the sharing channel are **red**: they rely on explicit user consent to export information and are at a higher risk of leaking private data. In Section 5.4, we describe how many top apps from the Google Play store fall into these categories.

2.6 Limitations and scope

πBox reduces privacy risks to the users of many apps and makes it more difficult to harvest large amounts of private user information, but it is not a privacy panacea.

First, the differentially private aggregate channel leaks a little information with every output. This is inevitable, and we quantify this leakage in Section 3. Note that no covert communication beyond this leakage is possible over the aggregate channel because differential privacy holds regardless of the recipient's auxiliary (including covert) information. In the case of πBox's aggregate channel, the *timing* of the release is differentially private, too, precluding a malicious app from encoding covert information in the timing of its aggregate outputs.

Second, while πBox's sharing channel guarantees that only the specified recipient can read the shared content, a malicious app may hide private information in this content via steganography. Several factors mitigate this risk. First, πBox shows the content to be shared to the user and uses the powerbox mechanism to directly confirm the user's consent to share. Second, πBox restricts the type of content to be shared: only plain text and images are allowed in our prototype.

This is a trade-off between usability and privacy. The design philosophy behind πBox is to avoid involving users in privacy-critical decisions (in contrast to the Android permission system). At the same time, sharing is important for many applications, and πBox lets users explicitly accept a privacy risk when sharing content.

Most importantly, πBox guarantees that shared content can only be viewed by the recipients who have been explicitly approved by the user. While a malicious app instance may be able to embarrass the user by sending private information to an approved recipient, the app publisher still does not have access to this data unless the recipient (or the user who is sharing) cooperates.

In general, we believe that πBox will be appealing to entities looking to (1) enhance or safeguard their existing app platforms by improving user privacy, (2) rent privacy-preserving cloud resources to app publishers, and/or (3) provide a curated version of their standard app store that offers privacy-enhanced apps to enterprise customers. πBox is an especially good fit for enterprise environments, where apps typically contain content from a single external publisher, do not require (in fact, frequently forbid) sharing of content outside the enterprise, do not rely on ads, and do not involve functionalities with multiple external parties such as brokered ad auctions.

3 Protecting privacy

The functionality of many apps depends, both technically and financially, on some information about their users. Aggregate statistics are often sufficient—for example, some ad-supported apps only need to track the number of ad impressions, not whether a particular user viewed a given ad—but even they may reveal information about individuals [8, 14].

πBox uses *differential privacy* [14] to enable app publishers to collect relatively accurate statistics on users' behavior while limiting information leaks about any individual user. Informally, differential privacy is a framework for designing computations where the influence of any single input on the output is bounded, regardless of the adversary's knowledge and/or external (auxiliary) sources of information the adversary may have access to.

"Conventional" differential privacy techniques such as the Laplacian mechanism (described in the following section) are primarily intended to protect individual inputs in computations on static datasets. By contrast, apps keep generating new data: for example, an app may continuously update the number of times a news article has been read or an ad has been shown. Moreover, app publishers may be interested in rankings, such as the most popular news articles or the most frequently misrecognized words in a transcription app. As we will show, conventional mechanisms, while privacy-preserving, result in an unacceptable loss of accuracy in these settings.

To balance privacy and accuracy, πBox deploys recently developed algorithms for differentially private counters under continual observation [16] and differentially private ranked lists [7]. To the best of our knowledge, πBox is the first system that uses differential privacy under continual observation in a working system.

3.1 Counters and top-K lists

In πBox, the key building block for the aggregate channel is a set of platform-controlled *counters*. As an app executes, it may increment one or more counters. Eventually, the (randomly perturbed) values of these counters are released to the app publisher. The list of counters must be defined by the publisher in advance. Therefore, a malicious app instance cannot encode user-specific information in its choice of counter *names*. The released counter *values* are differentially private and thus probabilistically hide the influence of any given user's data.

πBox enforces *user-level* differential privacy on these counters, i.e., the privacy of all data, actions, and any other inputs associated with a particular user, as opposed to the privacy of a single input. Formally, for some privacy parameter ϵ (described further in Section 3.2), a computation F satisfies user-level ϵ-differential privacy if, (1) for all input datasets D and D' that differ only in a single individual user whose inputs are present in D but not in D', and (2) all outputs $S \subseteq Range(F)$,

$$\Pr[F(D) \in S] \leq e^{\epsilon} \cdot \Pr[F(D') \in S] \qquad (1)$$

A standard mechanism for making any computation F differentially private is the *Laplacian mechanism*, which adds random noise from a Laplace distribution to the output of F before it is released, i.e., $F(x) + Lap\left(\frac{\Delta F}{\epsilon}\right)$. Here $Lap(y)$ is a Laplace-distributed random variable with mean 0 and scale y, and ΔF is the maximum possible change in the value of F (F's sensitivity) when a single user's inputs are removed from the dataset.

Intuitively, the more sensitive a computation is to its inputs, the more random noise is needed to ensure a given level of privacy. Consequently, ΔF in πBox—and, therefore, the amount of noise that πBox adds to the released counter values—depends on the number of counters a user can update (which we denote as n) and the maximum amount by which a user can affect any single counter (s). There is an important trade-off in the Laplacian mechanism between privacy (ϵ) and accuracy: *higher accuracy requires giving up more privacy*. We will revisit this trade-off in detail in Section 3.2.

Parameter chosen by platform provider
Per-period privacy budget (R)
Parameters chosen by app publisher
List of counters (L)
Frequency of output release (f)
Privacy parameter (ϵ)
Max. # counters app instance can update per period (n)
Max. contribution to each counter per period (s)
Buffer size (b)
of ranked counters (K)

TABLE 2—Parameters for aggregate counters. b and K only apply to delayed-output and top-K counters, respectively.

Supporting periodic updates. Many apps dynamically update counters during execution and then need to periodically release them. The Laplacian mechanism can be applied to every release, but if the timing of releases is independent of the counter's true value, the random noise added by the mechanism (which, too, is independent of the counter's value) can be much larger than the true value, resulting in high relative error. This arises, for instance, when counting the number of impressions for rarely displayed ads targeting a niche group of users.

πBox uses *delayed-output counters* [16] instead. Figure 3 describes how such a counter is implemented. Intuitively, this mechanism randomly delays releases of the counter value; if the value is small relative to the noise that must be added, the release is likely to be postponed.

Furthermore, rather than allowing counters to be continuously queried, πBox enforces a minimum interval between releases (line 5). Thus, even the counters that have internally accumulated a large number of updates may not be immediately released. Delaying the release may affect the freshness of the released values, but the relative error will be smaller.

Supporting ranked top-K lists. To release top-K lists, πBox adapts techniques by Bhaskar et al. [7]. The app publisher specifies K beforehand, and the amount of noise that is added is proportional to K, which is typically smaller than the amount of noise (proportional to n) that would have been added if we had used the Laplacian mechanism on every counter to determine the top K. To generate a ranking of the counters without their associated values, the algorithm adds $Lap(4Ks/\epsilon)$ random noise to the values of all counters and picks the

```
$V_i$ : true count in period $i$
$\lambda \Leftarrow \frac{s \cdot n}{\epsilon}$
$A \Leftarrow 0$
$D \Leftarrow b + Lap(\lambda)$
for each period $i$ of duration $1/f$ do
    $A \Leftarrow A + V_i$
    if $A - D > Lap(\lambda)$ then
        Release $A + Lap(\lambda)$
        $A \Leftarrow 0$
        $D \Leftarrow b + Lap(\lambda)$
    end if
end for
```

FIGURE 3—Delayed-output counter.

top K counters based on these noisy values. If an app publisher needs to know the actual values of the associated counters as well, the algorithm adds an additional $Lap(2Ks/\epsilon)$ noise to the true values of the selected K counters before releasing their values.

It may appear that the ability to release top-K lists allows apps to leak sensitive information. For example, the publisher of a password management app could learn the K most common user passwords (in any case, these are already well-known). Note, however, that the publisher cannot learn the password of any given user. Similarly, conventional differential privacy allows the publisher to ask how many users have a particular password, but the answer does not reveal any specific user's password.

Finally, πBox's aggregate channel can be extended to support other differentially private functions such as mean and threshold [48].

3.2 Choosing privacy parameters

Absolute privacy cannot be achieved: as long as the released values have any utility, the original data can be reconstructed after observing at most a linear (in the size of the dataset) number of values [13]. To model the cumulative loss of privacy after multiple computations on the same private data, differential privacy uses the notion of a *privacy budget* [15, 35]. Every ϵ-private computation charges ϵ cost to this budget. The higher the value of ϵ, the less noise is added, thus the released value is more accurate, but the privacy cost is correspondingly higher, too. The budget is pre-defined by the data owner. Once it is exhausted, no further release is allowed.

In our setting, it is undesirable for an app to lose functionality after a while. Instead, πBox enforces a *per-period* privacy budget that bounds *privacy loss per period* by parameter R, which is chosen by the platform provider. For a given R, the app publisher may specify the types of the counters the app will release (delayed-output and/or top-K with or without associated values), as well as the relevant parameters in Table 2, so long as

$$c \cdot f \leq R \qquad (2)$$

where $c = \epsilon/2$ for top-K counters without associated values and ϵ for the other two types of counters.

To understand how c and ϵ relate to the amount of information leaked, let P be an adversary's prior probability of the user's private data having a particular value and P' be the posterior probability after observing the released counters. Condition (1) ensures that $P' \leq e^c \cdot P$, i.e., any released value changes the adversary's prior probabilities (no matter what they are!) by no more than a constant multiplicative factor. If uncertainty is measured as min-entropy of the adversary's probability distribution over the private data,[4] every release yields $(c \log_2 e)$ bits of information to the adversary [1, 6]. Given this representation of uncertainty, πBox's counters release at most $(f \cdot c \log_2 e) = (R \log_2 e)$ bits per period. For example, an app that uses delayed-output counters with $\epsilon = 1$ and the release frequency f of once per day leaks at most 1.44 bits of information daily.

While it is straightforward to calculate how much noise should be added for a given choice of counter type and ϵ, the utility of a particular counter arguably depends not just on the amount of noise added, but also the actual true counter value, i.e., the *relative* amount of noise matters. The larger the true value, the larger the absolute noise that can be tolerated for a given relative error, thus allowing for smaller values of ϵ.

As long as condition (2) is followed, app publishers are free to choose the types of the counters used by their apps and the values of the parameters listed in Table 2. For example, a publisher may want more frequent output (f), at the expense of lower ϵ, higher λ and thus lower accuracy. To maintain the same accuracy, the publisher may keep the same λ at the cost of decreasing the maximum number of counters a single app instance can update (n) and/or the maximum amount it can contribute (s).

4 Implementation

We implemented a prototype of πBox using Android 2.3 (Gingerbread) for the device client; Jetty [29], a Java servlet container, for the remote services; and HBase [23] for the cloud communication and storage channels. The trusted computing base (TCB) consists of the above software, cloud operating system (Linux in our case), and the πBox implementation, which itself is approximately 7,500 lines of code for the cloud half and 2,700 for the device half. The design of πBox is largely agnostic to the specific sandboxing technology and could have used

[4] The min-entropy of a probability distribution that assigns probability p_i to some event i is $-(\max_i \log_2(p_i))$.

virtual machines, Native Client [39], or more advanced sandboxes, which would change the size of the TCB.

4.1 Isolation and authentication

Client isolation. To implement the sandbox on the device, we augmented Android's built-in sandboxing mechanism. By default, Android assigns each app a unique user identifier (UID). πBox allows non-privacy-preserving apps to coexist with privacy-preserving apps on the same device, but assigns UIDs from different ranges to apps of different types. This makes isolation enforcement simpler in the kernel code.

Android uses standard Linux permissions to isolate apps from each other, but this is not enough to prevent an app from abusing the permissions it has. To prevent πBox-confined apps from leaking private data, we modify Android to block them from creating world-readable files or directories, and from writing to files or directories owned by another app's UID.[5] πBox does not allow confined apps to communicate with other non-system apps via IPC, including Binder IPC (the basic primitive for various higher-level Android IPC mechanisms). Finally, we use iptables to confine the apps' network traffic. These changes are applied at the kernel level only to πBox-confined apps (recognized by their UIDs).

Cloud isolation. We implement the server-side functionality for πBox apps as Java servlets using Jetty. Many existing Web apps, e.g., those on Google App Engine [4], can thus be easily adapted to πBox.

In Jetty, each app is isolated in a separate Web app context (a container that shares the same Java class loader). In πBox, each user of an app is also isolated in a separate context, achieving classloader-level isolation. To restrict the servlet's communication via system resources, we rely on Java's security monitor. Our sandbox also includes many other restrictions used by Google App Engine, e.g., disallowing reflection and controlling access to JVM-wide resources such as system properties.

Authentication. When an app on the user's device wants to communicate with its cloud-based half, it sends an "intent" (a high-level IPC mechanism in Android) to πBox's local trusted authentication service, implemented as a system app. After identifying the requesting app, the authentication service requests the user's credentials via user input or from a cache and sends them, along with the app's ID, through a TLS tunnel to πBox's authentication manager in the cloud. Upon successful authentication, the authentication manager sets up a new servlet instance at a specific URL, establishes an IPsec endpoint on the machine where the servlet is instantiated, and sends this URL, a one-time password that is required to access the servlet instance, and the IPsec key to the authentication service on the user's device.

The authentication service establishes the other end of the IPsec tunnel on the device, updates iptables to allow the app to communicate with the servlet, and passes, via intent, the URL and password to the app. IPsec ensures that all communication to and from the servlet is encrypted, and iptables ensure that the app on the user's device can only communicate with the user's servlet instance via this IPsec tunnel. Finally, the app running locally on the user's device authenticates using the provided password via HTTP basic authentication over the IPsec tunnel (which encrypts the credentials); this step ensures that only this specific app can communicate with the servlet. Once this process is complete, the app can send HTTP requests to the provided URL and receive HTTP responses from its cloud component.

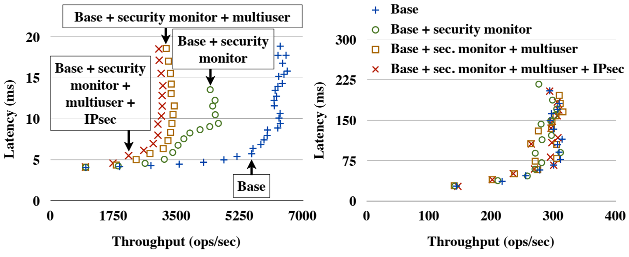
4.2 Storage and communication channels

πBox's storage systems use local device storage and HBase, a popular NoSQL storage system. Local device storage is part of πBox's private vault. Any data that is written to local storage is secured as described in Section 4.1 and cannot be exported from the sandbox. Access to cloud storage is provided via a HBase-like API.

When an app publisher submits an app to the platform, the publisher provides a WAR (Web application ARchive) file that contains the app's servlet code and XML files that describe the schemas of the HBase tables that the app needs for each type of cloud storage. To implement various channels, πBox provides wrappers of the HBase client that expose the appropriate interfaces to servlet instances. For example, the interface to content storage exposes read-only operations on the storage's shared tables. The interface to the cloud-backed private vault provides both read and write access to the per-sandbox table. The wrapper for the aggregate channel exposes an update-only interface for the counters, which are stored in the HBase tables by πBox. Stored counter values are periodically released by (1) sanitizing them via the differential privacy module using the parameters provided by the app publisher (Section 3.2) and (2) writing them to a table that can be read by the publisher.

The per-sandbox inbox allows a user's servlet to receive messages from the app publisher or from another user's servlet for the same app. This inbox is implemented using an HBase table in which each row corresponds to a single message. The row includes the sender's platform username (the name used to authenticate with the authentication service or a special username reserved for the app's publisher), a timestamp, and the message body. Messages from the publisher are de-

[5] This implies that an app can only write to directories that it alone has read access to and that other apps cannot see the files it has written.



(a) Static (b) SHA-256

FIGURE 4—Latency vs. throughput for πBox mechanisms.

livered to the recipient's inbox by a designated servlet, which can be invoked only by the authorized publisher.

Lastly, when an app wants to share content through the sharing channel, it sends an intent, along with the content to be shared, to πBox's sharing service, which is implemented as part of the authentication service. The sharing service prompts the user for the recipients' usernames and sends the message, along with the usernames of the sender and the recipients, to a designated servlet that only the platform can access. This servlet then adds the message to the inbox of each recipient.

5 Evaluation

5.1 Performance overhead

We evaluate πBox using a server with two four-core Xeon E5430 CPUs and 16 GB RAM and 4 clients with a single-core 3 GHz Pentium 4 Xeon CPU with hyper-threading and 1 GB of RAM, all running Fedora 8.

We first use micro-benchmarks to measure the throughput and response time of the various mechanisms employed by πBox on two types of workloads: a simple static workload where the server responds with about 10 bytes of static HTTP body data, and a computationally intensive workload where the server randomly generates 1 MB of data and calculates its SHA-256 hash. We generate the workloads by having a varying number of clients continuously submit requests over a 30-second interval.

Figure 4 shows the results with different components turned on. In the base configuration, we run the server with the Java security monitor disabled, no isolation (i.e., a single servlet instance serves all client requests), and without an IPsec tunnel between the server and the clients. We then enable the security monitor, run multiple servlet instances to serve different clients, and/or enable IPsec. For the simple static workload, πBox reduces the throughput of the system by roughly 50%, incurring an overhead of 0.17 ms per operation. For the heavier SHA-256 workload, however, the computation required to generate the hash effectively hides the overhead of πBox.

To measure the overhead of isolating app instances, we fix the load offered to the server (i.e., the number of requests generated by the clients) and vary the number of Web app containers (i.e., per-client servlet contexts) on the server. Figure 5 shows the throughput and response time of πBox for three types of workloads, with requests uniformly distributed across the containers. The static and SHA-256 workloads are the same as in the previous experiment. In the news reader workload, clients request a list of new articles (about 300) and a specific article (5 to 10 KB) from the servlet half of our news reader app (Section 5.3). This causes many I/O-intensive operations on the small HBase instance that stores the articles. As Figure 5 shows, the overhead of user isolation is insignificant for all three workload types.

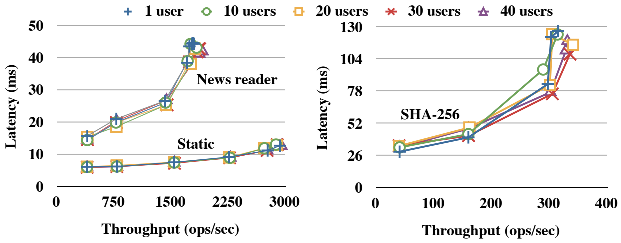
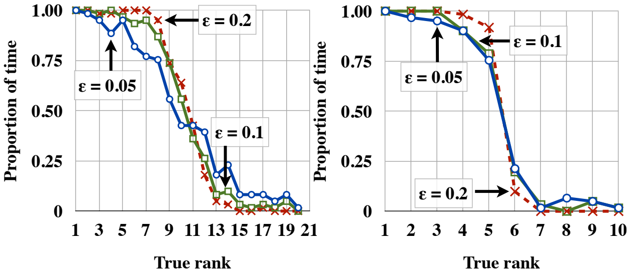


FIGURE 5—Overhead of user isolation for various workloads.



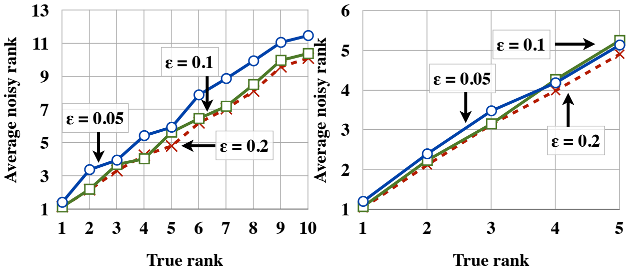
(a) Top-10 (b) Top-5

FIGURE 6—Fraction of time the true top-K documents appear in the noisy top-K list.

5.2 Privacy vs. accuracy

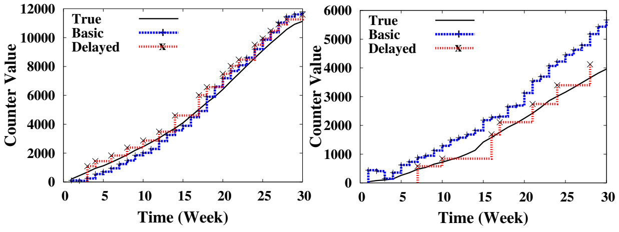
To show that the differential privacy mechanisms employed by πBox provide reasonable accuracy in real-world scenarios, we first apply the top-K mechanism to the 60-day Web server trace of the 1998 World Cup website [59]. For each day, we calculate the top 5 and top 10 most frequently accessed documents and use the πBox's aggregate channel to output "noisy" top-5 and top-10 lists. The total number of daily accesses for a top-10 document ranged from 6,000 to 14,000.

Figure 6 shows, as a function of the privacy parameter ϵ, how often the true top-5 and top-10 documents on a particular day appeared in the noisy, privacy-preserving top-5 and top-10 lists output by the aggregate channel. As ϵ increases, the accuracy of the noisy rank lists improves. For example, the 8th-ranked item appears in the



(a) Top-10 (b) Top-5

FIGURE 7—Average noisy rank for a given true rank.



(a) Frequently accessed (b) Infrequently accessed

FIGURE 8—Accuracy of delayed-output counter on two different documents. We use $\epsilon = 1$, $|L| = 100$, and $b = 500$.

noisy top-10 list 75% of the time when $\epsilon = 0.05$, but 95% of the time when $\epsilon = 0.2$. This percentage is even higher for items with higher true ranks. Figure 7 shows the average noisy rank given to true top-5 and top-10 documents. The accuracy of the noisy rank improves with higher ϵ; with $\epsilon = 0.2$, all ranks are correct.

To illustrate the advantages of the delayed-output mechanism for releasing infrequently updated counters, we use a trace from the University of Saskatchewan Web server [49] which contains a variety of access patterns. For this experiment, we set $\epsilon = 1$, the total number of delayed-output counters ($|L|$) to 100, the buffer size (b) to 500, and the release frequency to 1 week. We compare the delayed-output counter to a basic counter that simply outputs its differentially private value every week. Figure 8 shows the values of the delayed-output counter and the basic counter over a 30-week span for two documents with different access patterns. For the frequently accessed document, the delayed-output counter is off by 12.9% on average vs. 19.6% for the basic counter. For the less frequently accessed document, the delayed-output counter is much more accurate, with a relative error of 15.6% vs. 83.1% for the basic counter.

5.3 Apps

To illustrate how to build useful privacy-preserving apps in πBox, we developed three sample apps and ported two existing open-source apps.

Password manager. A password manager is an example of an app that needs to keep (but not share) sensitive data, e.g., store a user's credentials in the cloud so that the user can access them from different devices and to avoid keeping them on the devices themselves. Although many such apps use encryption, the user must trust that the app's publisher is neither malicious nor incompetent.

Our πBox-based password manager app simply stores the user's passwords in its cloud-backed private vault, enabling their retrieval from multiple devices. Despite its simple design, the app guarantees that (1) only a specific user can access the stored password via the app, and (2) the app cannot leak the stored passwords to anyone else (i.e., this app is "green"; see Section 2.5). This benefits both the user, who does not have to worry about the trustworthiness of the app, and the app publisher, who can rely on πBox to secure the publisher's app's storage.

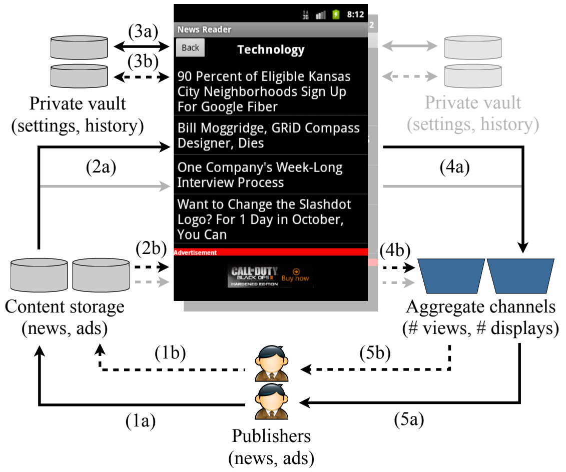


FIGURE 9—Interactions and data flow between the news reader app and πBox. The dark (solid, dotted) lines represent the flow from the (content, ad) publisher. The lighter lines represent the same flows for another user of the same app.

News reader. Our news reader app is an example of an ad-supported media browsing and consumption app that uses πBox's storage systems and involves multiple publishers. Figure 9 shows the flow of data between the publishers, the app, and the platform.

The main functionality in any news reader app is displaying content (news articles) to the user. In our implementation, the publisher supplies the articles by adding to, updating, and removing from the app's content storage located on πBox's cloud platform (Figure 9, 1a). The app has read-only access to this storage (Figure 9, 2a).

The news reader may provide personalized content to the user, for example, recommend certain articles based on the user's reading history. It can track the user's reading history by writing to its private vault (Figure 9, 3a). Because the vault is per-user and per-app, this data cannot leak to other app instances or the app publisher.

Many apps of this type are ad-supported. The ads may be published by either the app publisher or a separate entity, e.g., an advertising network partnering with the app publisher. Like the news articles, the ads are published and updated by their publisher and viewed by the user via content storage (Figure 9, 1b and 2b). Any personalization and micro-targeting is done by writing the relevant data to the private vault (Figure 9, 3b).

Both news and ad publishers may want to know how often their articles and ads have been viewed. Our app keeps one counter per article and ad. We use a top-10 list to track the most popular articles (Figure 9, 5a) and delayed-output counters for ad impressions (Figure 9, 5b), since the latter do not need to be released frequently.

The news reader app is a "yellow" app: although it exports statistics, πBox provides differential privacy guarantees to its users. It is straightforward to extend the news reader to let users share interesting articles with other users, which would make the app "red."

Transcription. Our transcription app uses cloud-based voice recognition. It records the user's speech on the device and transmits it to a servlet, which writes the recording to per-sandbox temporary scratch space and executes Sphinx-4 [53], an open-source speech recognition toolkit, to transcribe the text. The transcription is then sent back to and displayed by the app on the device. Our current prototype keeps the dictionary in the app's binary but we could also use content storage for this purpose, allowing the publisher to update the dictionary.

This app uses the aggregate channel to release the confidence scores of speech recognition for each l-gram ($l = 1$ in our prototype). First, the app publisher defines counters for all words in the Sphinx-4 dictionary (per Table 2, L is the list of these counters, $n = |L|$). Sphinx-4 provides confidence scores that range from 0 (low confidence) to 1. Because the publisher is likely interested in the most misrecognized words, our app inverts the score (thus making higher scores reflect lower confidence, up to a maximum of $s = 1$) before adding it to the previous value of the counter. The top-K list thus contains the K (10 in our prototype) most misrecognized words.

The transcription app is a "yellow" app. πBox guarantees that, even if the recordings of the user's speech contain highly sensitive data, the app can leak this data only through the differentially private aggregate channel as (noisy) top-K word lists, which do not identify the actual words spoken by specific users.

Porting existing apps. We ported OsmAnd [41], an Android navigation app based on OpenStreetMap [40], and ServeStream [51], an HTTP-streaming media player and media server browser, to πBox.

The major changes to the apps involved (1) adding code to initiate authentication via πBox's authentication service, (2) modifying all HTTP requests app to include the authentication credentials provided by the authentication service (Section 4.1), and (3) moving map and media content into πBox's content storage and serving them via servlets. The use of HTTP as the communication protocol simplified porting these apps to πBox, but this simplification is likely to apply to many other apps. Overall, for OsmAnd, we modified or added 174 out of 119,147 lines of code; for ServeStream, 133 out of 13,193 lines.

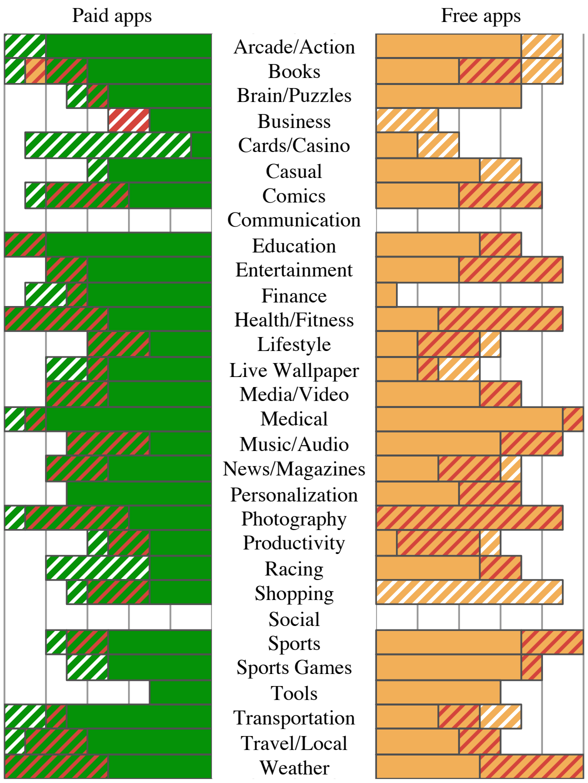


FIGURE 10—Number of top-10 apps in Google Play categories (as of Feb. 2013) that can be supported by πBox. Unsupported apps are uncolored/white. Stripes represent apps that, due to non-core sharing or unsupported functionality, are one color but whose core functionality is another color, e.g., a PDF viewer that allows sharing is red, but its core is green.

Both ported apps use only the private vault and content storage, making them "green."

5.4 Coverage of existing apps

To further evaluate how well πBox can support existing app functionalities, we surveyed the top 10 free apps and top 10 paid apps from all categories excluding wallpaper, widget, and library in the Google Play app store, for a total of 30 categories and 600 apps. This survey was based solely on the developer's description of the app in Google Play, thus the reported numbers are only estimates.

Figure 10 shows how many apps can be supported by πBox and the degree of support. Among the paid apps, 46% are green, 18% red, and 36% unsupported; considering only core functionality, 74% are green. Among the free apps, 37% are yellow, 21% red, and 42% unsupported; considering only core functionality, 67% are yellow. Unsurprisingly, many of the unsupported apps are those that are categorized as "communication" or "social" and thus require frequent sharing of data. Free apps are largely ad-supported and thus at least yellow.

6 Related work

xBook [52] and the system of Viswanath et al. [57] employ an extended sandbox mechanism similar to πBox for social-networking services. These systems protect user information stored on the platform (e.g., users' profiles and social relationships). Hails [21] protects user data on the platform using language-level information flow control. Unlike πBox, none of these systems can protect private information that apps directly receive or infer from their interactions with the users.

In xBook, each user decides whether to allow a particular domain to access a given part of the user's profile. By contrast, πBox simplifies users' decision-making by color-coding the apps based on their potential for privacy violations. xBook anonymizes app statistics (with no formal privacy guarantees), while the system of Viswanath et al. uses conventional differential privacy. As we show in Section 5.2, this can lead to high relative errors when releasing rarely updated values.

Embassies [26] is somewhat similar to πBox in that it aims to secure apps through a minimal interface that allows most apps to function correctly. Unlike in πBox, app publishers are not viewed as adversaries with respect to the user data collected by the app.

Dynamic taint analysis tracks the flow of sensitive data through program binaries [10, 24, 62] and can help protect user privacy. For example, TaintDroid [18] detects (rather than prevents) privacy violations, while AppFence [25] uses data shadowing and exfiltration blocking to prevent tainted data from leaving the device. Neither system handles implicit leaks. While taint-based systems can track specific data items such as device ID, they cannot prevent the app from leaking information about the user's behavior (e.g., articles the user has read). In general, dynamic taint tracking is complementary to the guarantees provided by πBox. For example, it can be used to prevent certain data items from being declassified even via differentially private channels.

Bubbles [55] aims to capture privacy intentions by clustering data into "bubbles" based on explicit user behavior. The privacy guarantee is similar to that of πBox's sharing channel: once the user adds a friend to a bubble, this friend gains access to all data in that bubble. Bubbles is limited to apps that run on the client device only.

ObliviAd [5] and PrivAd [22] are privacy-preserving online advertising systems that aim to protect user profiles from ad brokers. ObliviAd creates a black box at the ad broker using a secure coprocessor and oblivious RAM. This black box serves ads to clients, receives reports about ad clicks and impressions from clients via a secure TLS channel, records which ads were clicked or viewed (but not who viewed an ad), and only releases these records in large batches to make it difficult to determine who saw which ad. In PrivAd, clients fetch a large set of ads that are roughly based on users' interests; more accurate targeting is done only at the client. When the client reports which ads have been shown, a trusted third party anonymizes his identity before sending the data to the ad broker. By contrast, πBox aims to provide rigorous privacy guarantees without sacrificing the ability of advertisers to obtain accurate impression counts.

PINQ [35] and Airavat [48] are centralized platforms for differentially private computations on static datasets. PDDP [9] is a distributed differential privacy system in which participants maintain their own data.

While the cloud provider is trusted in πBox, CloudVisor [61] and CryptDB [44] focus on untrusted clouds. CloudVisor hides users' data from the hypervisor using nested virtualization, CryptDB uses encryption. CLAMP [42] employs isolation and authentication mechanisms that are similar to πBox to protect private data in LAMP-like Web servers. It focuses on compromised servers rather than malicious applications.

πBox can be viewed as imposing a mandatory information flow policy on untrusted apps. Previous work on information flow control includes [12, 32, 38, 60] and hundreds of other papers.

Bring-Your-Own-Device approaches that support dual workspaces [3, 58] enable personal and corporate data to coexist on the same device while permitting only trusted apps to access the corporate data. πBox takes this idea a step further and allows untrusted apps to run on corporate data, thus realizing the idea of Bring-Your-Own-App.

7 Conclusion

πBox is a new app platform that combines support for apps' functional needs with rigorous privacy protection for their users. Our evaluation demonstrates that πBox can be used in many practical scenarios, including "bring-your-own-app" enterprise deployments where external apps operate on proprietary company data.

Acknowledgments. This work was partially supported by the NSF grants CNS-0720649, CNS-0746888, CNS-0905602, CCF-1048269, and CNS-1223396, two Google research gifts, the MURI program under AFOSR grant FA9550-08-1-0352, and grant R01 LM011028-01 from the National Library of Medicine, NIH.

References

[1] M. S. Alvim, M. E. Andrés, K. Chatzikokolakis, P. Degano, and C. Palamidessi. Differential privacy: On the trade-off between utility and information leakage. *CoRR*, abs/1103.5188, 2011.

[2] Android malware promises video while stealing contacts. `http://blogs.mcafee.com/mcafee-labs/android-malware-promises-video-while-stealing-contacts`.

[3] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: A virtual mobile smartphone architecture. In *SOSP*, 2011.

[4] Google App Engine. `https://developers.google.com/appengine`.

[5] M. Backes, A. Kate, M. Maffei, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *S&P*, 2012.

[6] G. Barthe and B. Kopf. Information-theoretic bounds for differentially private mechanisms. In *CSF*, 2011.

[7] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *KDD*, 2010.

[8] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. "You might also like:" privacy risks of collaborative filtering. In *S&P*, 2011.

[9] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards statistical queries over distributed private user data. In *NSDI*, 2012.

[10] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum. Understanding data lifetime via whole system simulation. In *USENIX Security*, 2004.

[11] Cloud computing security policies, procedures lacking. `http://www.crn.com/news/security/224201359/cloud-computing-security-policies-procedures-lacking.htm`.

[12] D. E. Denning. A lattice model of secure information flow. *Communication of the ACM*, 19(5), May 1976.

[13] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

[14] C. Dwork. Differential privacy. In *ICALP*, 2006.

[15] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[16] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *STOC*, 2010.

[17] J. Edwards. How Microsoft's 'Do Not Track' policy is a mortal threat to ad exchanges like Facebook's. `http://www.businessinsider.com/microsofts-dnt-threatens-facebook-2012-9`.

[18] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.

[19] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *CCS 2011*.

[20] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *SOUPS*, 2012.

[21] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. Mitchell, and A. Russo. Hails: Protecting data privacy in untrusted web applications. In *OSDI*, 2012.

[22] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In *NSDI*, 2011.

[23] HBase. `http://hbase.apache.org`.

[24] A. Ho, M. Fetterman, C. Clark, A. Warfield, and S. Hand. Practical taint-based protection using demand emulation. In *EuroSys*, 2006.

[25] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In *CCS*, 2011.

[26] J. Howell, B. Parno, and J. R. Douceur. Embassies: Radically refactoring the web. In *NSDI*, 2013.

[27] W.-M. Hu. Reducing timing channels with fuzzy time. In *S&P*, 1991.

[28] IBM bans Dropbox, Siri and rival cloud tech at work. `http://www.theregister.co.uk/2012/05/25/ibm_bans_dropbox_siri`.

[29] Jetty. `http://www.eclipse.org/jetty`.

[30] A. Korolova. Privacy violations using microtargeted ads: A case study. *Journal of Privacy and Confidentiality*, 3(1), 2011.

[31] M. Krohn and E. Tromer. Noninterference for a practical DIFC-based operating system. In *S&P*, 2009.

[32] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard OS abstractions. In *SOSP*, 2007.

[33] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10), Oct. 1973.

[34] App sandbox in depth. `http://developer.apple.com/library/mac/#documentation/Security/Conceptual/AppSandboxDesignGuide/AppSandboxInDepth/AppSandboxInDepth.html`.

[35] F. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9), Sept. 2010.

[36] J. Mick. Android wallpaper app stole scores of users' data, sent it to China. `http://www.dailytech.com/Android+Wallpaper+App+Stole+Scores+of+Users+Data+Sent+it+to+China/article19200.htm`.

[37] S. Motiee, K. Hawkey, and K. Beznosov. Do Windows users follow the principle of least privilege? Investigating User Account Control practices. In *SOUPS*, 2010.

[38] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *SOSP*, 1997.

[39] NativeClient - native code for web apps. `http://code.google.com/p/nativeclient`.

[40] OpenStreetMap. `http://www.openstreetmap.org`.

[41] OsmAnd. `http://osmand.net`, `https://play.google.com/store/apps/details?id=net.osmand&hl=en`.

[42] B. Parno, J. M. McCune, D. Wendlandt, D. G. Andersen,

and A. Perrig. CLAMP: Practical prevention of large-scale data leaks. In *S&P*, 2009.

[43] iOS social apps leak contact data. `http://www.informationweek.com/news/security/privacy/232600490`.

[44] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *SOSP*, 2011.

[45] Real-time bidding in the United States and Western Europe, 2010-1015. `http://info.pubmatic.com/rs/pubmatic/images/IDC_Real-Time%20Bidding_US_Western%20Europe_Oct2011.pdf`.

[46] A. Reznichenko, S. Guha, and P. Francis. Auctions in do-not-track compliant internet advertising. In *CCS*, 2011.

[47] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *CCS*, 2009.

[48] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. In *NSDI*, 2010.

[49] Saskatchewan-HTTP - seven months of HTTP logs from the University of Saskatchewan WWW Server. `http://ita.ee.lbl.gov/html/contrib/Sask-HTTP.html`.

[50] M. Seaborn. Plash: Tools for practical least privilege. `http://plash.beasts.org`.

[51] ServeStream. `http://sourceforge.net/projects/servestream`, `https://play.google.com/store/apps/details?id=net.sourceforge.servestream&hl=en`.

[52] K. Singh, S. Bhola, and W. Lee. xBook: Redesigning privacy control in social networking platforms. In *USENIX Security*, 2009.

[53] CMU Sphinx - speech recognition toolkit. `http://cmusphinx.sourceforge.net`.

[54] 300,000 mobile apps stealing personal data. `http://www.itproportal.com/2010/07/29/300000-mobile-apps-stealing-personal-data`.

[55] M. Tiwari, P. Mohan, A. Osheroff, H. Alkaff, E. Shi, E. Love, D. Song, and K. Asanović. Context-centric security. In *HotSec*, 2012.

[56] Twitter apologizes for squirreling away iPhone user data. `http://www.csmonitor.com/Innovation/Horizons/2012/0216/Twitter-apologizes-for-squirreling-away-iPhone-user-data`.

[57] B. Viswanath, E. Kiciman, and S. Saroiu. Keeping information safe from social networking apps. In *WOSN*, 2012.

[58] VMware Horizon Mobile. `https://blogs.vmware.com/euc/2012/08/vmware-horizon-mobile-on-ios.html`.

[59] 1998 World Cup website access logs. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`.

[60] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in HiStar. In *OSDI*, 2006.

[61] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *SOSP*, 2011.

[62] Q. Zhang, J. McCullough, J. Ma, N. Schear, M. Vrable, A. Vahdat, A. C. Snoeren, G. M. Voelker, and S. Savage. Neon: System support for derived data management. In *VEE*, 2010.